DIGITAL INDUSTRIES SOFTWARE

Software threats only come in one size: devastating

Identifying, addressing and defending against cyber threats with proactive strategies

Executive summary
Software vulnerabilities are much like an iceberg. Like the tip of an iceberg, a software development team has great visibility into the software source code the team is developing. Like an iceberg, though, much of any software product is not visible to the software development team. The development team has complete visibility into the code they author. Beyond the source code managed by the software development team, typical software builds include source code and objects that come from external libraries, open-source projects and third-party vendors. Much of this is opaque to the software product development team.

Kevin Sweeney and Tom Behrens, Siemens Digital Industries Software

SIEMENS

Introduction

Recently, the vulnerabilities surrounding Log4J have placed information technology (IT) organizations on edge as they search for where vulnerable versions of it are embedded in their software. According to Infosecurity Magazine, in 2020 there were an average of 119,000 cyber-threats per minute to IT systems.

A key component for protecting an organization from vulnerabilities is ready access to the components that make up the software. This ready access can be provided by maintaining a software bill-of-materials (SBOM) for all of the software that an organization builds. As new threats are discovered, a simple lookup against the organizations' SBOMs can reveal if vulnerabilities raised by these threats exist in the software. Due to the nature of software threats and vulnerabilities, speed of detection is extremely important in preventing potential catastrophe.

What is a bill-of-materials?

Product development organizations have maintained various types of documentation regarding bill-of-materials (BOM) for their products. Oftentimes, the company will generate the BOM in conjunction with computer-aided design (CAD) tools and product lifecycle management (PLM) software. A BOM structure and contents will vary depending on its application. For example, a single product may have an engineering bill-of-materials (EBOM) that is used in designing the product and a manufacturing bill-of-materials (MBOM) intended for the factory to build the product. These may be single-level lists of the components that make up a product, or they may be hierarchical representations of a product.

Software is often represented as a single component in an EBOM or MBOM without any breakdown of how that software is built. Software is increasingly a key part of today's products, so it is now more important than ever to know how it is built. Unfortunately, for a traditional hardware product a BOM does not provide sufficient detail to track and manage threats and vulnerabilities in a software program. The answer to this shortfall is to generate an SBOM as the software is built. This software bill-of-materials can be used standalone or incorporated into a traditional product BOM.

Why would you use a SBOM?

The most prominent use case for an SBOM is to manage threats and vulnerabilities. The SBOM can be interrogated against vulnerability databases to quickly identify threats. Using the Log4J example, wouldn't it be great to push a button and immediately understand if there is exposure, and if so, where that exposure is? Having an SBOM is a powerful tool in license compliance, especially for open-source components that make up your software products.

Software quality assurance is dependent on the knowledge of what a software is program built on. Increasingly, regulators, business partners and customers are demanding auditability from software developers. Siemens Digital Industries Software believes the SBOM should be generated in multiple, user-selectable industry standard formats from the development database containing meta data that answers a variety of questions on the development of software products with the touch of a button.

Concept of the SBOM

An SBOM should contain not only meta data regarding the source code the organization authors and compiles, but also should contain meta data on library artifacts, open-source components, utility components and third-party software that goes into the built software. In addition, the SBOM must detail both the environment in which the software was built and where it is targeted for deployment.

For example, software built on top of one version of Linux may have different vulnerability profiles than software built with the same source code on a different version of Linux. For this reason, the SBOM must identify all of the components that make up a software build and the environment in which that software was built. The SBOM identifies and lists the software components within a built software product including important meta data on each component and the structure of the relationships between those components. Earlier, we mentioned that a BOM may be flat or hierarchical. An SBOM must be hierarchical to show the relationships between components.

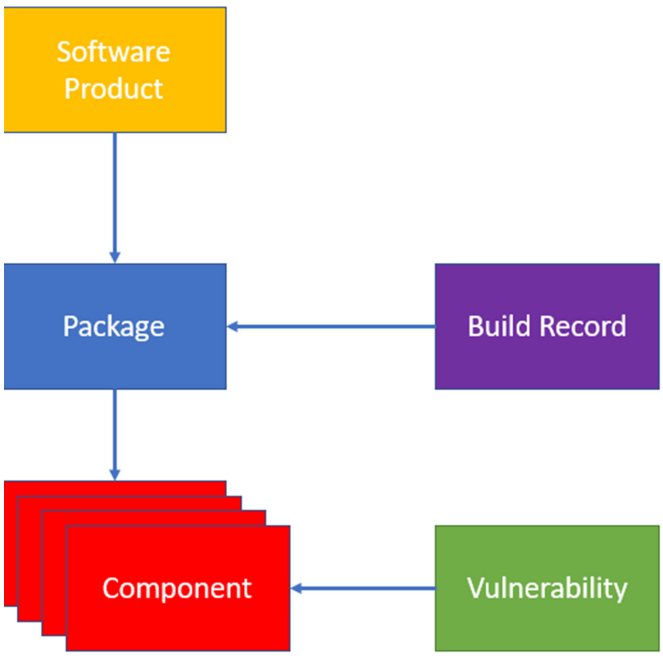

Figure 1. The four primary data model elements.

Requirements for a SBOM

SBOMs can be created internally as part of the internal development processes or other product vendors may provide them. This white paper focuses on creating SBOMs as an integral part of those internal product development processes. It is also possible to utilize hybrid processes, where the generation of internal SBOMs is further enhanced by the inclusion of vendor-supplied SBOMs for products that are included in the internal builds of software. This last point illustrates the value of identifying, if possible, whether a component is at a root level, or if it has other "child" components. Often this is not known and will be noted on the affected components.

An SBOM must faithfully reflect all components that went into a software build. The components included in the SBOM must identify the version or build of each component. As an example, at the time of this writing, serious cybersecurity threats have been identified for specific versions of Log4J. In this case, it is not sufficient to simply identify that a software product contains Log4J. It is now necessary to know which version of Log4J was injected into a software product.

The generation of an SBOM should be automated as much as possible to ensure accuracy and prevent human error. There are multiple automation points in the generation of an SBOM. The first automation point occurs with the software build. As an example, a process workflow can trigger the initiation of a software build with Jenkins. Jenkins is an open-source software tool for automated testing. An SBOM management solution should be configured to parse the build results to create the meta data for the build.

In the diagram below, we can see there are four primary data model elements. A package work item will be created for each build. It will act as the root or top level for the meta data surrounding each software build. The build record will be automatically created for each Jenkins build by the SBOM solution. The build record will supply the implementation environment information to supplement the SBOM details that will be captured in the package and component work item types. The component items will be created via scripts in the SBOM solution. The component items will have a one-to-one correspondence to the artifacts that are included in a build. The component items will be in a hierarchical structure that will store all of the meta data for generating an SBOM.

Although the package and component items will contain the data needed to generate an SBOM, they will not represent the final SBOM structure. Generating the SBOM from the package and component items provide flexibility. The SBOM solution will allow the user to generate SBOMs to match scenario-based standards. This flexibility helps the SBOM solution to generate SBOMs for the Software Package Data Exchange (SPDX) standard or the CycloneDX standard depending on the need of downstream users.

Flexibility in representation provides the capability for the SBOM solution to interface with vulnerability databases at the individual work item level or by using either of the standard SBOM formats. Currently, the expectation is the SBOM solution will have been integrated with external vulnerability databases. Typically, it is not necessary to maintain vulnerability databases since these are readily available from other sources.

The SBOM must contain an overview of the built software product and details of its components. Per the above diagram, the information that drives the generation of the SBOM is contained in the package, build record and component entities.

A package will be generated for each build of the application. It will document a high-level overview of the built application including important identifying values such as a unique application version name and a checksum or hash that can be used to verify the software on disk, in fact, is the software described in the SBOM. Structurally, the package is the parent container for the SBOM.

The build record will be generated as an effect of the software build. It documents the environment in which the build was executed. The version of the operating system (OS), as an example, may have a significant impact on the resulting build. A build of an application on top of Red Hat Enterprise Linux 8 may have significant differences, including vulnerabilities, from the same application generated on top of SUSE Enterprise Linux 15. These differences extend to specific builds of the operating system so that is tracked as well. The same concerns apply to the choice of compilers, so this information is also documented in the build record. The date timestamp acts as an identifier for the build and is tracked along with the build master or person responsible for the software product build.

The details of the SBOM are provided in the component items. At a minimum, there will be a component item for each item included in the build. This information will automatically be created by integrating with the external build platform (which could be Jenkins, for example). The component will provide identifying information such as the version of the component, checksum or hash of the component, license information if applicable and most important, the name(s) that will be used for lookup of the component in external vulnerability databases. Because software is built in layers, the component has a flag to indicate whether it is a root or atomic item. If it is not a root item, the assumption is the component is built of other components.

SBOM standards

There are multiple standards for the SBOM. The primary standards are SPDX and CycloneDX. There is also a third standard that does not work as well with managing software threats and vulnerabilities called software identification (SWID) tags. For the SBOM solution, the focus should be primarily on SPDX and CycloneDX. Both can be targeted while generating SBOMs from the meta data model documented in this paper.

SPDX

SPDX is an open standard for producing a SBOM. It was created by the Linux Foundation with the intent of facilitating license management in an open software stack. SPDX or Software Package Data Exchange has subsequently been extended to support management of software vulnerabilities. SPDX has been submitted to and accepted by the International Organization for Standardization (ISO) 5962. SPDX supports multiple formats including SPDX/XML, JSON, YAML, RDF/XML and Microsoft XLS.

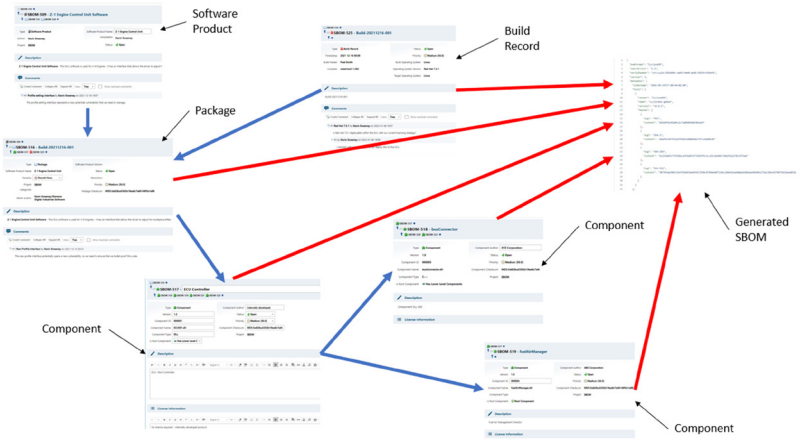

Figure 2. In the diagram above, the blue lines show the connections between the meta data element types. The red lines show the inputs for SBOM generation.

CycloneDX

CycloneDX is a standard supported by the Open Web Application Security Project (OWASP). Although CycloneDX is much newer than SPDX, it appears to have strong momentum in the software vulnerabilities space. CycloneDX supports SBOM in both XML and JSON formats.

Relationship from meta data to SBOM

The following diagram depicts the relationship between the meta data described in the previous section and the generated SBOM. The separation of the meta data from the SBOM provides a number of significant values.

A key value of maintaining the meta data as input to the SBOM is that a single set of meta data can be used to generate SBOMs to multiple standards. A software vendor may want to share SBOMs with customers that have different requirements. If a customer requires the SPDX standard, a SPDX standard SBOM can be generated. Similarly, they can generate a CycloneDX standard SBOM using the same meta data.

In addition to providing support for the SBOM, the meta data can support other software management use cases. This serves as a complete guide to your built software. The SBOM is intended to be machine-readable, while the meta data can be queried and reported on in a variety of user-defined formats.

SBOM and vulnerability databases

The SBOM must constantly be compared against one or more vulnerability databases (VDB). Identifying and tracking threats and vulnerabilities in a database is a large and complex effort that is often managed by specialist organizations. The foundation of successful vulnerability management is a comprehensive VDB that is continuously updated. Organizations cannot protect themselves against vulnerabilities they don't know about.

The U.S. government funds a VDB called common vulnerabilities and exposures (CVE). This is maintained by the MITRE Corporation. This is commonly used as a base input to other more advanced VDBs. The official U.S. government VDB, the National Vulnerability Database (or NVD), is updated from the CVE with a medium time lag of seven days according to a study published by the threat intelligence firm Recorded Future in June 2017. This time lag between publishing to CVE and NVD creates a window of opportunity to exploit a vulnerability. For all VDBs, a critical consideration is reducing the time between discovery of a vulnerability and publishing it in the VDB.

A variety of commercial and propriety VDBs often extend the government vulnerability databases. Examples of these include IBM's X-Force and the Open-Sourced Vulnerability Database (OSVDB). Large organizations such as Siemens will often supplement external VDBs with its own internal proprietary VDBs. One of the key advantages of supplemental internal VDBs is the ability to rapidly update potential vulnerabilities on a proactive basis while others may still be analyzing and determining whether to update a VDB.

Timing is critical for comparing a SBOM and vulnerability databases. Any latency between a vulnerability addition to a VDB and its discovery in the SBOM increases the potential risk to the organization. Ideally, the interaction between the VDB and the SBOM is interactive, but if that is not possible, interactions should be scheduled with the greatest practical frequency. Vulnerabilities in the SBOM are likely to remain hidden until the SBOM is compared with the VDB.

Closing the loop with ALM

Generation of the SBOM is incorporated into the continuous integration/continuous deployment (CI/CD) pipeline that is managed as an integral process of the application lifecycle management (ALM) domain. ALM processes offer comprehensive tracking across the lifecycle.

Source code artifacts that are included in a build are traced to the requirements the source code implements. When the CI/CD pipeline verifies an integration of a new build, test cases from the ALM domain are incorporated to complete the loop on traceability from requirements to test cases that verify the requirements to built artifacts that implement the requirements.

Managing software threats and vulnerabilities includes the need to identify risks and develop requirements for risk mitigation. The requirements for risk management are traced directly to the meta data elements that drive SBOM creation. This traceability, combined with verification against a vulnerability database, creates a closed-loop solution for software threat and vulnerability management.

Traceability between the implemented SBOM, source code, requirements, test cases and risk management are essential for meeting regulatory requirements across industries sensitive to risk and safety mitigation. These industries include aerospace, automotive, cybersecurity, defense, electric distribution grid, medical devices and more. As an example, DO178C provides guidance on software considerations for airborne software systems certification. Implementing DO178C requires traceability from requirements to specific source code lines that implement the requirements. Similar upstream and downstream traceability requirements exist across other regulated industries, such as ISO 26262 for automotive and IEC 62304 for medical device software that can also benefit from creation of an SBOM.

Conclusion

Constructing the SBOM in parallel with the software development process and within an existing application lifecycle management tool that organizes and manages that process does several things.

First, it reduces the extra time it takes to gather the specific elements of the SBOM and organize them into a meaningful and orderly document. Second, any software improvements, updates, bug fixes and the like are immediately incorporated into the SBOM as those changes happen. This results in a more accurate and up-to-date SBOM that can be released as the new software.

Lastly, the SBOM becomes the core tool for understanding the internal structure of developed software products. It provides the foundation for identifying threats and vulnerabilities that may be incorporated into software products by constantly comparing SBOMs with published vulnerability databases. In virtually every industry, identifying and responding to threats and vulnerabilities in software products is an essential capability. Failure to have this capability can have catastrophic consequences for those who create, use and are peripherally impacted by software products, including software vendors, customers and the public at large.

References

1. 119,000 Threats Per Minute Detected in 2020 Infosecurity Magazine article on cyber threats detected in 2020.
   https://www.infosecurity-magazine.com/news/119k-threats-per-minute-detected/
2. Energy SBOM Proof of Concept
   https://inl.gov/sbom-poc/
3. SBOM Energy Webinar
   https://www.youtube.com/watch?v=2zGMr47Dvuo
4. NTIA SBOM Proceedings
   https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
5. Minimum Elements for a SBOM from NTIA:
   https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
6. NTIA SBOM Reference List:
   https://ntia.gov/SBOM
7. NIST Responsibilities to Executive Order 14028:
   https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity
8. Executive Order 14028
   Executive Order on Improving the Nation's Cybersecurity
   https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Siemens Digital Industries Software

Americas: 1 800 498 5351

EMEA: 00 800 70002222

Asia-Pacific: 001 800 03061910

For additional numbers, click here.

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Xcelerator, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

siemens.com/software

84375-D2 03/22 K